Trial-based Heuristic Tree Search for MDPs with Factored Action Spaces

Florian Geißer
The Australian National University
florian.geisser@anu.edu.au

David Speck
University of Freiburg
speckd@informatik.uni-freiburg.de

Thomas Keller
University of Basel
tho.keller@unibas.ch

Abstract

MDPs with factored action spaces, i.e. where actions are described as assignments to a set of action variables, allow reasoning over action variables instead of action states, yet most algorithms only consider a grounded action representation. This includes algorithms that are instantiations of the trial-based heuristic tree search (THTS) framework, such as AO*or UCT.

To be able to reason over factored action spaces, we propose a generalisation of THTS where nodes that branch over all applicable actions are replaced with subtrees that consist of nodes that represent the decision for a single action variable. We show that many THTS algorithms retain their theoretical properties under the generalised framework, and show how to approximate any state-action heuristic to a heuristic for partial action assignments. This allows to guide a UCT variant that is able to create exponentially fewer nodes than the same algorithm that considers ground actions. An empirical evaluation on the benchmark set of the probabilistic track of the latest International Planning Competition validates the benefits of the approach.

Introduction

Markov decision processes (MDPs) allow to model probabilistic decision making problems. Factored MDPs represent the state and action space compactly by describing the semantics of the MDP in terms of state and action variables. Popular algorithms to solve MDPs are tree search algorithms such as UCT (Kocsis and Szepesvári 2006) or AlphaGo Zero (Silver et al. 2017), or heuristic search algorithms such as AO*(Nilsson 1980). Trial-based heuristic tree search (THTS) (Keller and Helmert 2013) allows to model a broad family of algorithms under one common framework by specifying six components. A THTS algorithm gradually builds up the AND/OR tree that is induced by the underlying MDP in a sequence of trials to determine an estimation of the expected value of the initial state of the MDP. THTS algorithms are therefore well-suited for anytime optimal planning, as they are able to provide a recommendation on which decision to take at any point.

MDPs with factored action spaces have recently seen renewed interest, as many applications can be described within this framework, such as satellite mission planning (Povéda et al. 2019), conversation planning (Xue, Fern, and Sheldon 2014), or the automatic construction of operation policies for dam management (Reyes et al. 2015). Ontañón (2013) and Moraes et al. (2018) investigated Monte Carlo tree search for combinatorial multi-armed bandits by dividing them into a collection of multiple traditional multi-arm bandits. However, THTS algorithms do not explicitly consider a factored action representation. This becomes a problem when the number of action variables increases, as the number of actions grows exponentially and with this the branching factor of the MDP. Moreover, many THTS algorithms, including UCT, require to simulate each decision at least once, which becomes infeasible when facing exponentially many actions.

In this work, we consider *domain-independent* THTS algorithms for factored MDPs and focus on factored action spaces. We represent decisions in the search tree as *decision trees over action variables*, where each outcome corresponds to an assignment of an action variable. The key insight is that the factored action representation allows search algorithms to concentrate on promising assignments of actions without spending effort on unfruitful paths. To be able to initialise decision nodes in the factored decision tree we generalise state-action heuristics to heuristics that allow the assignment of estimates for partial action variable assignments, and show how to decompose any state-action heuristic to a heuristic that can be applied to the factored action representation. We present a theoretical evaluation which shows that many common components of THTS preserve their behaviour in the factored action representation. This motivates us to extend the THTS implementation of the PROST planning system (Keller and Eyerich 2012) with a factored action representation. We evaluate the approach on the benchmark set of the past probabilistic tracks of the International Planning Competition which includes problems that allow the concurrent application of multiple actions. In addition, we investigate mutex detection in order to combine multiple binary action variables into one finite-domain action variable. In cases where all actions can be combined into one finite-domain action variable our approach is equivalent to the original flat action representation, so a factored action representation is only considered when necessary. The evaluation shows that this representation can strengthen the current state of the art in probabilistic planning.

Background

An MDP (Puterman 1994) is a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, where $\mathcal{S}$ is a finite set of states, $\mathcal{A}$ is a finite set of actions, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and the transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ defines the probability $\mathcal{P}(s'|s, a)$ that applying action a in state s leads to state s'. We define the set of successors of state s and action a as $\mathsf{succ}(s, a) = \{s' \in \mathcal{S} | \mathcal{P}(s'|s, a) > 0\}$. We say action a is applicable in state s iff $\mathsf{succ}(s, a) \neq \emptyset$ and denote the set of applicable actions in s as $\mathcal{A}(s)$.

We consider *finite-horizon MDPs*, where the number of action applications is limited by the horizon $H \in \mathbb{N}$, and we augment the state space such that the number of remaining steps is part of a state, denoted by $s[h]$ for $s \in \mathcal{S}$. We have $\mathcal{P}(s'|s, a) = 0$ if $s[h] \neq s'[h - 1]$, to enforce that the number of remaining steps decreases by one in each transition. We further assume that $\mathcal{A}(s) \neq \emptyset$ for all $s \in \mathcal{S}$ to ensure that there are no dead-ends. A *terminal state* is a state with $s[h] = 0$, and s_I specifies the *initial state*. Therefore, the finite-horizon MDP $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, H, s_I \rangle$ induces a directed acyclic graph.

In this work, we focus on MDPs where the action space $\mathcal{A}$ is induced from a set of action variables, also called *base actions* B, where each $\mathsf{b} \in \mathsf{B}$ is associated with a finite domain $\mathcal{D}_\mathsf{b} = \{0, \ldots, |\mathcal{D}_\mathsf{b}| - 1\}$. A *partial action assignment* is a partial function $a : \mathsf{B} \hookrightarrow \bigcup_{\mathsf{b} \in \mathsf{B}} \mathcal{D}_\mathsf{b}$, such that $a(\mathsf{b}) \in \mathcal{D}_\mathsf{b}$ for all $\mathsf{b} \in \mathsf{B}$ where a is defined. We sometimes abuse notation and write (b, k) instead of $a(\mathsf{b}) = k$, when a will be clear from context. For binary domain values we also sometimes write $\bar{\mathsf{b}}$ for $(\mathsf{b}, 0)$ and b for $(\mathsf{b}, 1)$. We denote the base actions for which a is defined as $\mathsf{basis}(a)$. If a assigns a value to each $\mathsf{b} \in \mathsf{B}$ we call a an *action state*, or simply *action*. The set of all actions is $\mathcal{A}$ and corresponds to the previous notion of actions in the MDP.

A solution to an MDP is a policy $\pi : \mathcal{S} \to \mathcal{A}$ s.t. $\pi(s) \in \mathcal{A}(s)$, i.e. a mapping from states to applicable actions. The *expected reward* of π is given by the *state-value function* $V^\pi(\mathcal{M}) = V^\pi(s_I)$ with

$$V^\pi(s) = \begin{cases} 0 & \text{if } s \text{ is terminal} \\ Q^\pi(s, \pi(s)) & \text{otherwise} \end{cases}$$

where $Q^\pi(s, a) = \mathcal{R}(s, a) + \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \cdot V^\pi(s')$ is the *action-value function*. An optimal policy π^* is a solution to the well-known Bellman optimality equation (Bellman 1957):

$$V^*(s) = \begin{cases} 0 & \text{if } s \text{ is terminal} \\ \max_{a \in \mathcal{A}} Q^*(s, a) & \text{otherwise,} \end{cases}$$

$$Q^* = \mathcal{R}(s, a) + \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \cdot V^*(s').$$

Trial-based Heuristic Tree Search

We focus on online algorithms which interleave planning and execution. The THTS framework (Keller and Helmert 2013; Keller 2015) is a generalisation of Monte Carlo tree search which allows to describe online algorithms by specifying different components. The core of every THTS algorithm is an explicit search tree of *decision* and *chance* nodes. A decision node is a tuple $n_d = \langle s, V^k \rangle$, where $s \in \mathcal{S}$ is a state and $V^k \in \mathbb{R}$ is the state-value estimate based on the first k trials. A chance node is a tuple $n_c = \langle s, a, Q^k \rangle$, where $s \in \mathcal{S}$ is a state, $a \in \mathcal{A}$ is an action and $Q^k \in \mathbb{R}$ is the action-value estimate based on the first k trials. We refer to the separate components of a decision node n_d as $s(n_d)$, $V^k(n_d)$, and of a chance node n_c as $s(n_c)$, $a(n_c)$ and $Q^k(n_c)$.

Initially the search tree only contains the root decision node n_0 for the initial state s_I. As the name indicates, THTS algorithms are trial-based, and each trial consists of multiple phases: in the selection phase, the explicit tree is traversed by choosing successor nodes according to *action selection* and *outcome selection* components until a previously unvisited decision node is reached. This initiates the expansion phase, where for each action a child chance node is added to the tree, initialised with a heuristic value based on the applied *initialisation function*. As a result, all successor nodes of the currently visited decision node have action-value estimates and the selection phase starts again. The *trial length* component determines when a trial ends. Then, the value of all nodes which were visited in the current trial are updated in reverse order, based on the *backup function*. A trial is completed after the backup function is called on the root node. A new trial is initiated, unless some constraint (e.g. time limit or limit on the number of trials) is violated. Finally, the *action recommendation function* chooses which action will be executed. Algorithm 1, adapted from Keller (2015) depicts the skeleton of a THTS algorithm.

Algorithm 1: The THTS schema.

```
1  Function THTS(M, timeout-constraint):
2      n_0 = getRootNode(M)
3      while moreTrials(n_0) do
4          visitDecisionNode(n_0)
5      return recommendAction(n_0)
6  Function visitDecisionNode(n_d):
7      if n_d was never visited then
8          initialiseNode(n_d)
9      n_c = selectAction(n_d)
10     visitChanceNode(n_c)
11     backupDecisionNode(n_d)
12 Function visitChanceNode(n_c):
13     if trialEnds(n_c) then
14         n_d = selectOutcome(n_c)
15         visitDecisionNode(n_d)
16     backupChanceNode(n_c)
```

Depending on the underlying components, search nodes can have additional annotations. Given a node n, we denote the number of times it has been selected at the end of the k-th trial as $\mathcal{L}^k(n)$.

To motivate the next section we will look at a small MDP and consider the UCT algorithm which aims to balance the exploitation of known good decisions and the exploration of potentially better actions by application of UCB1 action selection (Auer, Cesa-Bianchi, and Fischer 2002): given a

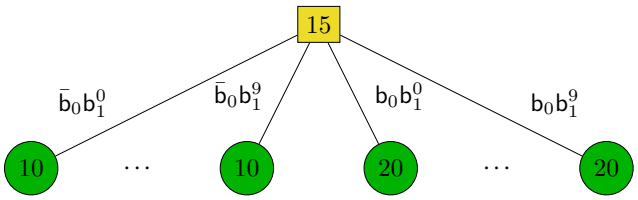

Figure 1: Search tree of a multi-armed bandit problem with flattened action representation. Decision nodes are yellow squares, and chance nodes are green circles. Nodes are annotated with their value estimates, and b_1^i in actions is short for (b_1, i).

decision node n_d, UCT chooses the successor node n_c that maximises the formula $\mathcal{B} \cdot \sqrt{\frac{\log \mathcal{L}^k(n_d)}{\mathcal{L}^k(n_c)}} + Q^k(n_c)$, where $\mathcal{B} \in \mathbb{R}^+$ is a bias parameter. In case there is a successor node with $\mathcal{L}^k(n_c) = 0$ this node will be selected instead. UCT performs Monte Carlo backups to update the state-value and action-value estimates of visited nodes with the following function:

$$V^k(n_d) = \begin{cases} 0 & \text{if } s(n_d)[h] = 0 \\ \frac{\sum_{n_c \in \mathsf{succ}(n_d)} \mathcal{L}^k(n_c) \cdot Q^k(n_c)}{\mathcal{L}^k(n_d)} & \text{otherwise,} \end{cases}$$

$$Q^k(n_c) = \mathcal{R}(n_c) + \frac{\sum_{n_d \in \mathsf{succ}(n_c)} \mathcal{L}^k(n_d) \cdot V^k(n_d)}{\mathcal{L}^k(n_c)},$$

where $\mathsf{succ}(n)$ is the set of successor nodes of node n and $\mathcal{R}(n_c) := \mathcal{R}(s(n_c), a(n_c))$.

Running Example

Consider an MDP with a single non-terminal state, horizon 1, and 20 possible action states which can be decomposed into two base actions b_0 and b_1 with $\mathcal{D}_{\mathsf{b}_0} = \{0, 1\}$ and $\mathcal{D}_{\mathsf{b}_1} = \{0, ..., 9\}$. The reward is additively decomposable such that any action state that contains $(\mathsf{b}_0, 0)$ has a reward of 10 and actions with $(\mathsf{b}_0, 1)$ have a reward of 20. In other words, to select the best action it is sufficient to reason over b_0, since b_1 is not important for the final outcome.

Figure 1 depicts the search tree after 20 trials, i.e., after each action has been selected exactly once (if no heuristic is used for node initialisation, each action has to be selected at least once). Observe that the state-value estimate of the root node (15) is far off from the optimal value estimate (20), since all sub-optimal children had to be selected at least once. While additional trials will from now on only select actions where $\mathsf{b}_0 = 1$, the necessary initial selection of sub-optimal actions has a negative effect on the value estimation of the root node: even after 1000 trials the state-value estimate is only 19.9 (and each of the 10 action states containing $\mathsf{b}_0 = 1$ was visited 99 times).[1]

The example reveals two potential pitfalls: 1) The number of action states is exponential in the number of base actions. Even if only a subset of base actions is relevant for achieving good results exponentially many actions have to be selected at least once before we can follow promising paths. 2) The outcome of sub-optimal actions can affect the state estimate over a large number of visits. This is particularly important when we have to consider anytime algorithms for large MDPs, where the number of trials is often limited by tight time limits and it is important to quickly discover promising paths.

[1] We consider a bias parameter of $\mathcal{B} = 1$ for this example.

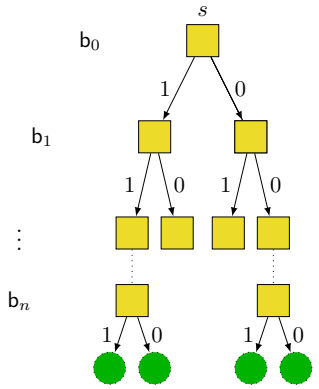

Figure 2: Example for a factored decision node tree. Each layer corresponds to the assignment of a single base action.

The question is how to make use of the factored action representation and focus on the relevant part of the search space. The original idea for our approach comes from Keller and Eyerich (2012), who consider probabilistic planning problems with exponentially many probabilistic outcomes, but a transition function with the property that the transition function can be decomposed into transition functions over state variables. They exploit this by representing a chance node with multiple layers of nodes, one node for each state variable. They note that this technique can also be used for concurrent actions, but the problems they consider have only a low grade of concurrency and therefore only small action spaces. In this work, we extend the THTS framework and consider a factored action tree representation instead of single decision nodes.

THTS for Factored Action Spaces

To be able to represent factored action states as a decision tree we have to adapt the definition of decision nodes. The definition of chance nodes continues to be as before. A decision node is now a tuple $n_d = \langle s, \hat{a}, \mathsf{b}, Q^k_{\hat{a}} \rangle$, where $s \in \mathcal{S}$ is a state, $\hat{a}$ is a partial action assignment, b is a base action, and $Q^k_{\hat{a}}$ is the value estimate of this node. We refer to the separate components of n_d as $s(n_d), a(n_d), \mathsf{b}(n_d)$ and $Q^k(n_d)$. We can interpret $Q^k_{\hat{a}}$ as the *partial action-value estimate* of state s where the assignment of base actions in $\mathsf{basis}(\hat{a})$ is already fixed, but we have a choice for the assignment of the remaining base actions. In the following, we assume an *ordering* $o : \mathsf{B} \to \{0, \ldots, |\mathsf{B}| - 1\}$ on the base actions, which determines on which base action to branch next. Given a state s, the root node of a factored decision tree is $n_d = \langle s, \emptyset, \mathsf{b}, Q^k_\emptyset \rangle$ where b is the base action with $o(\mathsf{b}) = 0$ and $Q^k_\emptyset = V^k$. Consider Figure 2, where we have $\mathsf{B} = \{\mathsf{b}_0, \ldots, \mathsf{b}_n\}$ with binary domains for all b_i and $o(\mathsf{b}_i) = i$ for $i = 0, \ldots, n$.

Algorithm 2: Visiting factored decision nodes.

```
1  Function visitDecisionNode(n_d = ⟨s, â, b, Q^k_â⟩):
2      if n_d was never visited then
3          initialiseNode(n_d)
4      a = â ∪ selectAssignment(n_d)
5      if |a| = |B| then
6          visitChanceNode(⟨s, a, Q^k⟩)
7      else
8          b' ← arg min_{b'∈B\basis(a)} o(b')
9          visitDecisionNode(⟨s, a, b', Q^k_a⟩)
10     backupDecisionNode(n_d)
11 Function initialiseNode(n_d = ⟨s, â, b, Q^k_â⟩):
12     for d ∈ D_b do
13         a = â ∪ (b, d)
14         if a is inconsistent with A(s) then
15             continue
16         if |a| = |B| then
17             addChanceNode(s, a, h(s, a))
18         else
19             b' ← arg min_{b'∈B\basis(a)} o(b')
20             addDecisionNode(s, a, b', h(s, a))
```

Given a decision node $n_d = \langle s, \hat{a}, \mathsf{b}, Q^k_{\hat{a}} \rangle$, we refer as before to the set of successor nodes as $\mathsf{succ}(n_d)$, but this can now be either a set of decision nodes or a set of chance nodes. To explicitly denote the chance nodes resulting from following successors of n_d we write $\mathsf{succ}_c(n_d)$. Note that the set of actions corresponding to $\mathsf{succ}_c(n_d)$ is $\{a \in \mathcal{A}(s(n_d)) | \hat{a}(n_d) \subset a\}$, i.e. actions that are applicable in the state corresponding to n_d and consistent with the partial action assignment $\hat{a}$. From now on we write *factored representation* when we consider THTS based on a decision tree representation and *flattened representation* otherwise.

To be able to perform trial-based heuristic tree search with decision trees instead of single decision nodes we have to adapt how decision nodes are visited. Algorithm 2 shows the adaptation of the corresponding part of the original THTS algorithm. To initialise nodes (line 11) we must check whether the base action assignment is consistent with the applicable actions $\mathcal{A}(s)$. If this is the case and the assignment results in an action state we create a chance node with initial estimate $h(s, a)$. If a is only a partial action state we create a new decision node with heuristic value $h(s, a)$ for the base action b' according to ordering o. Heuristic functions required for the initialisation of partial action states are described in the next section.

If n_d was already visited we require a function that allows the selection of action assignments (line 4). When we only consider an assignment to the current base action b the selection function can be implemented as in the original THTS, as we simply have to select one of the children of n_d. More sophisticated selection functions that assign multiple base actions at once are possible, but not the focus of this work. Once we have a complete assignment of base actions (line 5), the next child will be a chance node and the THTS scheme is as before. Otherwise we continue with the next decision node, dictated by the ordering on the base actions.

The second component of THTS that has to be adapted is the recommendation of the final action. Instead of recommending the action of the most promising decision node, the function must now recursively select the most promising base action assignment. The final set of base actions then corresponds to the action that is recommended by the algorithm.

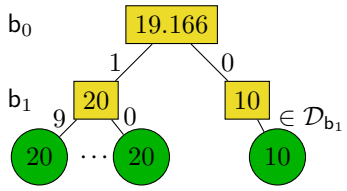

Figure 3: Search tree of a multi-armed bandit problem based on factored representation.

Running Example

Figure 3 shows the resulting search tree with underlying base action ordering $o(\mathsf{b}_i) = i$ for $i \in \{0, 1\}$ of the running example after 11 trials. At this point action selection will select one of the actions where $\mathsf{b}_0 = 1$ a second time. There are a couple of interesting observations. First, the state estimate at the root node is already more informed after the first 10 trials compared to the flattened representation. When we investigate how search performs with additional trials we see that after already 100 trials the state estimate of the root node reaches 19.9 (compared to 1000 trials with a flattened representation). Second, we visit a sub-optimal action only once, all other sub-optimal actions are not even part of the current search tree. This is particularly important as the number of actions grows exponentially with the number of base actions: in the case of binary base actions THTS with a flattened representation is required to initialise potentially up to $2^{|\mathsf{B}|}$ decision nodes, whereas the factored representation only requires $2 \cdot |\mathsf{B}|$ node initialisations.

The curious reader might wonder how the algorithm performs when we choose a bad ordering, i.e. the unimportant base action b_1 is considered first. In this case the first layer consists of 10 decision nodes, one for each value of b_1. Initially, all these nodes have to be selected at least once, and for each node two successors are initialised (for the values of b_0). Thus, after 10 trials the complete tree is initialised, consisting of 10 decision nodes and 20 chance nodes, and the initial root estimate is 10. The behaviour for subsequent trials depends on the tie-breaking strategy of the selection function. In the worst case each of the 10 initial decision nodes is selected a second time. In each case one of the children corresponding to the values of b_0 has already been selected, so the other value is chosen. Thus, after 20 trials we have visited every chance node at least once and get the same behaviour as with flattened representation.

Factored Heuristics

Classical probabilistic planning heuristic functions assign a numerical value to each state or to each state-action pair. As the number of successor states of applying an action in a state can be as high as the number of states, *state-action heuristics* $h : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ are often preferable. However, to initialise decision nodes in a factored representation we must be able to compute partial action value estimates. Let $\hat{\mathsf{B}}$ be the set of partial action states over B (including proper action states). We extend state-action heuristics to *factored heuristics* $\hat{h} : \mathcal{S} \times \hat{\mathcal{A}} \to \mathbb{R}$ which assign a value to pairs that consist of a state $s \in \mathcal{S}$ and a possibly partial assignment $\hat{a} \in \hat{\mathcal{A}}$ to the action variables. We propose three factored heuristics that have in common that they are parameterised by a given state-action heuristic.

Definition 1. Let $h : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a state-action heuristic, s a state and $\hat{a}$ a partial action state. The *factored-max heuristic* h^{fm}_h of h is

$$h^{\text{fm}}_h(s, \hat{a}) = \max_{\{a' \in \mathcal{A}(s) \mid \hat{a} \subseteq a'\}} h(s, a')$$

The factored-max heuristic assigns the highest heuristic value of an applicable action in s that is consistent with $\hat{a}$. When $\hat{a}$ is a full assignment $\hat{a}$ is also the only applicable action consistent with itself, thus h^{fm}_h corresponds to h on full action assignments.

Theorem 1. *Let $s \in S$ be a state, $a \in \mathcal{A}(s)$ be an action and h be a state-action heuristic. Then $h^{fm}_h(s, a) = h(s, a)$.*

Proof. As a is an action it is also a full action assignment and hence $\{a' \in \mathcal{A}(s) \mid a \subseteq a'\} = \{a\}$. □

Theorem 1 allows to compare THTS configurations based on a flattened representation with heuristic h to THTS configurations based on a factored representation with heuristic h^{fm}_h, as full action states are initialised with the same values, and differences between the algorithms are hence only due to the different structures of the search space.

The second factored heuristic we consider, the *decomposed heuristic* h^{dc}_{h,a_0}, exploits the factored form of action states to invoke a given state-action heuristic h only on a subset of the action states. The heuristic considers the base action assignments as features of action states and computes *feature weights* for all pairs (b, k) where $\mathsf{b} \in \mathsf{B}$ and $k \in \mathcal{D}_{\mathsf{b}}$ in each state. It derives a heuristic estimate for a partial action state $\hat{a}$ by summing up all feature weights of $\hat{a}$. This idea has been applied before to approximate the value of a state with factored value functions (Koller and Parr 1999; Guestrin et al. 2003) or potential heuristics (Pommerening et al. 2015), but, to the best of our knowledge, not in the context of factored actions and with entirely different methods to obtain feature weights.

In addition to the state-action heuristic h, the decomposed heuristic takes a *reference action* a_0 as input. In a state s, the weight of feature (b, k) is computed as

$$w_{(\mathsf{b},k)} := \frac{h(s, a_0)}{|\mathsf{B}|}$$

if it is a feature of a_0 i.e., we assume that each feature of a_0 has the same influence on $h(s, a_0)$. While this might not always be a reasonable assumption, we leave it for future work to look into a better suited distribution of $h(s, a_0)$ over the features of a_0.

To obtain the weight for a feature (b, k) that is not a feature of a_0, we consider the action state a' that is equal to a except that $(\mathsf{b}, a_0(\mathsf{b}))$ is replaced by (b, k), i.e. $a' = a_0 \setminus \{(\mathsf{b}, a_0(\mathsf{b}))\} \cup \{(\mathsf{b}, k)\}$ and compute the weight as

$$w_{(\mathsf{b},k)} := h(s, a') - \frac{|\mathsf{B}| - 1}{|\mathsf{B}|} \cdot h(s, a_0).$$

The weights of $|\mathsf{B}| - 1$ many features of a', all except for (b, k), are determined from a_0. Thus, $w_{(\mathsf{b},k)}$ is chosen such that the weights of all features in a' sum up to $h(s, a')$.

Definition 2. Let $h : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a state-action heuristic, a_0 a reference action, s a state and $\hat{a}$ a partial action state.

The *decomposed heuristic* h^{dc}_{h,a_0} of h and a_0 is

$$h^{\text{dc}}_{h,a_0}(s, \hat{a}) = \sum_{(\mathsf{b},k) \in \hat{a}} w_{(\mathsf{b},k)},$$

where the weights $w_{(\mathsf{b},k)}$ are computed as described above.

The decomposed heuristic computes feature weights in a way that $h^{\text{dc}}_{h,a_0}(s, a) = h(s, a)$ if $a = a_0$ or if a is one of the a' that are used to compute feature weights. As equivalence is not guaranteed for other action states, we cannot present a result analogous to Theorem 1 for the decomposed heuristic.

However, an advantage of h^{dc} is the potentially smaller number of calls to the state-action heuristic h.

Theorem 2. *Let $s \in S$ be a state and $\hat{a}$ a partial action. Computing $h^{fm}_h(s, \hat{a})$ requires up to exponentially more computations of h than computing $h^{dc}_{h,a_0}(s, \hat{a})$.*

Proof sketch. If all action assignments are applicable in s, the number of invocations of h to compute $h^{\text{fm}}_h(s, \hat{a})$ is exponential in the number of unassigned base actions. If $\hat{a}$ is a full action assignment such that $\hat{a}(\mathsf{b}) \neq a_0(\mathsf{b})$ for all $\mathsf{b} \in \mathsf{B}$, h^{dc}_{h,a_0} invokes h once with a_0 and once for every base action, i.e. a number that is linear in the number of base actions. □

The linear dependency on the number of action fluents is an important advantage of h^{dc}, in particular in planning tasks with a large number of applicable actions. To assess the impact of this property empirically, we propose a third heuristic. The *factored-random heuristic* h^{fr}_h is a variant of h^{fm} that samples a random applicable action state a that is consistent with $\hat{a}$ and computes $h^{\text{fr}}_h(s, \hat{a})$ as $h(s, a)$. Like h^{fm}, h^{fr} computes a heuristic solely by considering (one) consistent and applicable action state(s), and like h^{dc}, the number of invocations of h is subexponential in the number of action fluents.

Definition 3. Let $h : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ be a state-action heuristic, s a state and $\hat{a}$ a partial action state.

The *factored-random heuristic* h^{fr}_h of h is

$$h^{\text{fr}}_h(s, \hat{a}) = h(s, a),$$

where a is sampled uniformly at random from the set of full action assignments that are applicable in s and consistent with $\hat{a}$.

Theoretical Evaluation

In the following, we show that many component instantiations of the original THTS framework are preserved under the factored representation, i.e. their behaviour does not change. We will discuss backup functions, their properties in relation to UCB1 action selection, and recommendation functions. Outcome selection components require no adaptation, as the definition of chance nodes does not change. Trial length components that condition on decision nodes only require a small adaptation, such that these conditions now only trigger on decision nodes where $\hat{a} = \emptyset$.

In the following we assume w.l.o.g that base actions have binary domains and the ordering on base actions is $o(\mathsf{b}_i) = i$ for $\mathsf{B} = \{\mathsf{b}_1, \ldots, \mathsf{b}_n\}$.

Backup Functions

We start our theoretical evaluation with the backup function component. Recall the definition of Monte Carlo backups. With a factored representation we slightly have to adapt the backup function of a decision node. Instead of updating state-value estimates V^k, we now update partial action-value estimates $Q^k_{\hat{a}}$ and assume that $Q^k_\emptyset = V^k$. Since successors of decision nodes can now be chance nodes (if $\hat{a}$ is a full assignment) or other decision nodes (otherwise), the backup function for decision nodes needs to be adapted in the latter case by using decision node successors instead of chance node successors on the right hand side of the equation.

In the following we show that the backup function leads to the same state value estimate V^k, regardless of whether we consider a factored or a flattened representation, with the condition that the chance nodes of which we backup values have the same annotations (i.e. selection count and action-value estimate).

Theorem 3. *Let $\mathcal{M}$ be an MDP with factored action space base action set B, s a state, $n_d = \langle s, V^k \rangle$ a decision node in the flattened representation, and $n'_d = \langle s, \emptyset, \mathsf{b}, Q^k_\emptyset \rangle$ a node in the factored representation. If $\mathsf{succ}(n_d) = \mathsf{succ}_c(n'_d)$ then Monte Carlo backup guarantees $V^k(n_d) = Q^k_\emptyset(n'_d)$.*

Proof Sketch. The proof is by induction over $|\mathsf{B}|$. In the base case there is only a single base action, and therefore $\mathsf{succ}_c(n'_d) = \mathsf{succ}(n'_d) = \mathsf{succ}(n_d)$, i.e. both decision nodes have the same successors and therefore backup assigns the same value estimate. For the inductive case we consider n'_d and there are two successors, one for each value of b_1: $n'_{\bar{\mathsf{b}}_1}$ and n_{b_1}, and we have

$$Q^k_\emptyset(n'_d) = \frac{\mathcal{L}^k(n'_{\bar{\mathsf{b}}_1}) \cdot Q^k_{\bar{\mathsf{b}}_1} + \mathcal{L}^k(n'_{\mathsf{b}_1}) \cdot Q^k_{\mathsf{b}_1}}{\mathcal{L}^k(n'_{\bar{\mathsf{b}}_1}) + \mathcal{L}^k(n'_{\mathsf{b}_1})}.$$

The key insight is that $\mathcal{L}^k(n'_{\bar{\mathsf{b}}_1})$ is the annihilator for the denominator in

$$Q^k_{\bar{\mathsf{b}}_1} = \frac{\mathcal{L}^k(n'_{\bar{\mathsf{b}}_1\bar{\mathsf{b}}_2}) \cdot Q^k_{\bar{\mathsf{b}}_1\bar{\mathsf{b}}_2} + \mathcal{L}^k(n'_{\bar{\mathsf{b}}_1\mathsf{b}_2}) \cdot Q^k_{\bar{\mathsf{b}}_1\mathsf{b}_2}}{\mathcal{L}^k(n'_{\bar{\mathsf{b}}_1\bar{\mathsf{b}}_2}) + \mathcal{L}^k(n'_{\bar{\mathsf{b}}_1\mathsf{b}_2})},$$

since $\mathcal{L}^k(n'_{\bar{\mathsf{b}}_1}) = \mathcal{L}^k(n'_{\bar{\mathsf{b}}_1\bar{\mathsf{b}}_2}) + \mathcal{L}^k(n'_{\bar{\mathsf{b}}_1\mathsf{b}_2})$. With a similar argument for n'_{b_1} we then get $Q^k_\emptyset(n'_d) = \frac{\sum_{a \in \hat{\mathsf{B}}} \mathcal{L}^k(n'_a) \cdot Q^k_a}{\mathcal{L}^k n'_d}$, where $\hat{\mathsf{B}} = \{\bar{\mathsf{b}}_1\bar{\mathsf{b}}_2, \mathsf{b}_1\bar{\mathsf{b}}_2, \bar{\mathsf{b}}_1\mathsf{b}_2, \mathsf{b}_1\mathsf{b}_2\}$. Thus, the value estimation of n'_d sums over all chance nodes, and the result is exactly the same as in the case of flattened representation. □

Theorem 3 guarantees that the value estimate of the root decision node in the factored representation is equal to the root decision node in the flattened representation, but only if corresponding chance nodes have been visited an equal amount of times and were assigned the same value estimates. We therefore also consider a second backup function which does not count how often a node has been selected and instead exploits the declarative model of the MDP.

Partial Bellman backups (Keller and Helmert 2013) are a variation of Full Bellman backups and consider the probabilities of outcomes when computing action value estimates. In their original form they are given as

$$V^k(n_d) = \begin{cases} 0 & \text{if } s \text{ is terminal} \\ \max_{n_c \in \mathsf{succ}(n_d)} Q^k(n_c) & \text{otherwise,} \end{cases}$$

$$Q^k(n_c) = \mathcal{R}(n_c) + \frac{\sum_{n_d \in \mathsf{succ}(n_c)} \mathcal{P}(n_d|n_c) \cdot V^k(n_d)}{\mathcal{P}^k(n_c)}.$$

Here, $\mathcal{P}(n_d|n_c) = \mathcal{P}(s(n_d)|s(n_c), a(n_c))$, and $\mathcal{P}^k(n_c)$ is the sum of probabilities of all outcomes that are explicit in the k-th trial, i.e. $\mathcal{P}^k(n_c) = \sum_{n_d \in \mathsf{succ}(n_c)} \mathcal{P}(n_d|n_c)$.

Partial Bellman backups allow to label nodes as solved, which indicates that the optimal value of the corresponding state (resp. action) is known. This can be exploited by algorithms, since subtrees of such nodes do not have to be visited again.

When considering a factored representation we have to make the same adaptation as we had to with Monte Carlo backups: successors of decision nodes can be decision nodes again, and we update partial action-value estimates. Then, this backup function behaves identically as in the case of flattened representation.

Theorem 4. *Let $\mathcal{M}$ be an MDP with factored action space with base action set B, s a state, $n_d = \langle s, V^k \rangle$ a decision node in the flattened representation, and $n'_d = \langle s, \emptyset, \mathsf{b}, Q^k_\emptyset \rangle$ a node in the factored representation. If $\mathsf{succ}(n_d) = \mathsf{succ}_c(n'_d)$ then Partial Bellman backup guarantees $V^k(n_d) = Q^k_\emptyset(n'_d)$.*

Proof. Bellman backups calculate $Q^k_\emptyset(n'_d)$ by recursively selecting the maximum of all children, until a chance node is selected. Thus, $Q^k_\emptyset(n'_d) = \max_{n_c \in \mathsf{succ}_c(n'_d)} Q^k(n_c) = \max_{n_c \in \mathsf{succ}(n_d)} Q^k(n_c) = V^k(n_d)$. □

If the underlying selection strategy explores the whole tree, Partial Bellman backups converge towards the Bellman optimality equation (Keller and Helmert 2013). With a flattened representation Monte Carlo backups converge towards the optimal value function if for a node n_d the outcome selection component guarantees that $\frac{\mathcal{L}^k(n_d)}{\mathcal{L}^k(n_c)} \to \mathcal{P}(n_d|n_c)$ for $k \to \infty$ and the action selection strategy guarantees that $\frac{\mathcal{L}^k(n^*_c)}{\mathcal{L}^k(n_d)} \to 1$ for $k \to \infty$, where $n^*_c = \langle s, \pi^*(s), Q^k \rangle$ is

the successor of n_d in the optimal policy π^* (Keller and Helmert 2013).[2] This is the case for UCB1 action selection and, with a similar argument to that of Keller and Helmert (2013), holds also in the factored representation, as all outcomes will be sampled proportional to their probability, and UCB1 never stops exploring.

While we do not discuss other backup functions in detail we mention that Theorem 4 also holds if we consider Max-Monte-Carlo backups (Keller and Helmert 2013) or Full Bellman backups.

Recommendation Function

Formally, a recommendation function maps the search tree resulting from k trials to a probability distribution over $\mathcal{A}(s_I)$, the applicable actions of the initial state. Keller (2015) considers the following functions: 1) The *max child* recommendation (Chaslot et al. 2008; Bubeck, Munos, and Stoltz 2009), also known as expected best arm recommendation (EBA) which recommends the action with the highest value estimate, and 2) The *most played arm* recommendation (Chaslot et al. 2008; Bubeck, Munos, and Stoltz 2009) that recommends the action that was selected most often. In both cases ties are broken uniformly at random. Since both functions aggregate the maximum over all children they result in the same action recommendation when we consider a factored representation. The proof is analogous to the proof of Theorem 4.

Empirical Evaluation

While the theoretical evaluation shows that the factored representation preserves many properties of THTS components it does not reveal whether the representation results in stronger algorithms. We therefore evaluate our approach on the benchmark set of the probabilistic track of the previous three international planning competitions (IPPC). While benchmarks of the IPPC 2011 and 2014 rarely contain concurrently applicable actions, a recent analysis indicates that the problems with factored action spaces and large sets of concurrently applicable actions that can be found among the IPPC 2018 domains pose a major challenge for current planners (Geißer, Speck, and Keller 2019). Such actions can be understood as base actions b with binary domain, where $a(\mathsf{b}) = 1$ corresponds to the application of b.

Our implementation is based on the PROST planning system (Keller and Eyerich 2012), which was the winner of the IPPC 2011 and 2014 and serves as the baseline for PROST-DD (Geißer and Speck 2018), the winner of the IPPC 2018. All algorithms we consider share the following component configuration: a trial stops at the end of the horizon, action selection is based on UCB1, outcome selection performs Monte Carlo sampling, the backup function applies partial Bellman backups, EBA is the recommendation function and the state-action heuristic is the iterative deepening search (IDS) component of PROST. We allow a time window of 1 second per planning step. In each step, the planner submits an action for the current state and receives a successor state from the rddlsim simulator (Sanner 2010). A planning round is finished once the end of the horizon is reached. To obtain statistically significant results we perform 100 planning rounds and average the accumulated reward. All experiments have a memory limit of 4GB and share a cluster of Intel Xeon Silver 4114 2.2 GHz machines. We measure the quality of each configuration with the IPC score, which assigns a score of 1 to the best configuration, a score of 0 to a very simple baseline policy or every configuration that is worse than this policy, and a linearly interpolated value in between these extremes to the remaining configurations. In cases where the expected reward of an optimal policy is unknown, the IPC score is not an optimal measure of performance (Geißer, Speck, and Keller 2019), but it still indicates relative planner performance and is easier to represent than absolute average rewards. The code, benchmarks and data set are available at `https://doi.org/10.5281/zenodo.3749869`.

Mutex Invariants Since the IPPC benchmark sets of 2011 and 2014 only rarely contain concurrently applicable actions the implementation of a factored representation may introduce unnecessary overhead in such cases. To alleviate this, we compute *mutual exclusion* (*mutex*) invariants as known from classical planning (Helmert 2009). Two binary action variables are called *mutex* if there exists no applicable action state that assigns the value 1 to two base actions. Action variables $\mathsf{b}_1, \ldots, \mathsf{b}_n$ that are pairwise mutex can be transformed into a *single* common action variable b with domain $\mathcal{D}_\mathsf{b} = \{0, 1, \ldots, n\}$, where $a(\mathsf{b}) = 0$ encodes a state where all b_i for $1 \leq i \leq n$ are assigned a value of 0 and $a(\mathsf{b}) = i$ corresponds to the action state only containing $(\mathsf{b}_i, 1)$. As a consequence, we can perform a mutex detection algorithm in a precomputation step that allows us to transform base actions that are not concurrently applicable into a single base action. Note that in cases where all original base actions can be transformed into a single base action, the factored representation corresponds to the flattened representation, as we have a single action variable with one domain value per action state. Of the 280 instances in our benchmark set, 120 fall in this category.

In the following, we investigate two questions: 1) To what extent can the ordering on base actions influence planning performance, and 2) How does the factored representation compare to the flattened representation under the same configuration setting.

Base Action Orderings

In our first experiment we want to see how different orderings affect the accumulated average reward. For this, we consider h^{dc} as underlying heuristic and compare 25 random variable orderings[3] on the ACADEMIC ADVISING 2018 domain, which is the domain that has the highest number of concurrently applicable actions among all domains.

Figure 4 depicts the Gaussian Distributions of the average rewards. We only report problems where a non-trivial policy

[2] Assuming w.l.o.g. that there exists only one optimal policy.

[3] We performed a second experiment with 100 random orderings, but on a different machine. The outcome of the experiment did not change, so we do not include this in our final report.

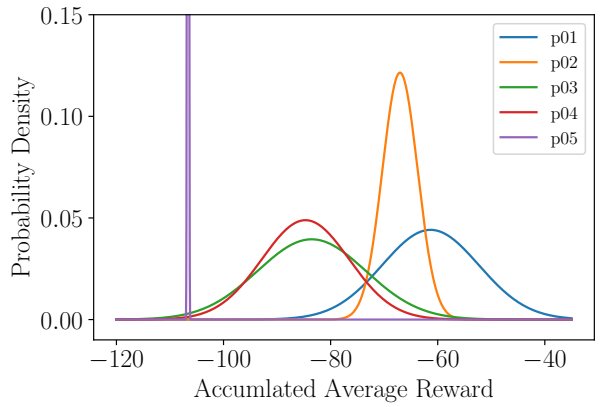

Figure 4: Density Functions of the Gaussian Distributions representing the accumulated average rewards for different problems of ACADEMIC ADVISING 2018 for configurations based on random variable orders without mutex precomputation.

Domain	**INC**	**DEC**	h^{dc}	h^{fm}	h^{fr}	**flat**
AA'14 (10)	2.93	1.65	2.95	2.95	2.82	**3.98**
AA'18 (20)	2.42	2.63	4.23	4.87	5.27	**5.52**
CHROM. (20)	19.15	17.46	**19.32**	18.18	19.20	16.99
COOP. RECON (20)	5.59	9.66	**12.07**	11.38	11.21	12.00
CROSS. (10)	**9.90**	9.45	9.34	9.42	6.47	9.44
EARTH (20)	13.85	18.17	17.83	18.00	**19.19**	17.64
ELEVATORS (10)	8.96	9.03	9.60	9.57	8.72	**9.80**
GAME (10)	8.22	8.18	**9.81**	9.73	9.68	**9.81**
LUCK (20)	12.26	12.58	13.05	16.66	14.40	**17.40**
MAN. (20)	3.72	5.43	5.93	4.43	**8.24**	4.11
NAV. (10)	9.04	9.44	**9.63**	9.22	8.65	9.42
RECON (10)	8.78	9.75	9.73	9.74	7.72	**9.93**
RFBE (20)	9.66	11.42	12.19	12.31	11.35	**12.82**
SKILL (10)	9.17	8.88	**9.86**	9.72	9.37	9.66
SYSADMIN (10)	6.82	6.75	8.01	7.74	7.88	**9.93**
TAMARISK (10)	8.58	7.69	9.57	9.69	8.06	**9.80**
TRAFFIC (10)	9.59	9.54	9.72	9.40	9.56	**9.85**
TRIANGLE (10)	4.33	4.05	6.10	4.19	2.27	**7.42**
WILDFIRE (10)	6.51	3.73	7.69	**7.70**	5.49	7.45
WILDLIFE (20)	12.43	11.81	15.41	15.05	**17.26**	10.89
Sum (280)	171.90	177.30	202.02	199.95	192.81	**203.85**

Table 1: IPC scores over all problem instances and configurations. Number of instances is denoted in brackets. The first two columns show the incremental and decremental action variable ordering without mutex detection and based on h^{dc}. The remaining columns show the result of the different heuristics with activated mutex detection against the flattened representation.

is executed. The remaining cases are too hard for our configurations, which therefore execute a policy that is no better than the simple baseline policy independent of the ordering. While the impact of the ordering is negligible in p05, it has a strong impact in p01, p03 and p04, and a still significant impact on p02, indicating that a good ordering plays an important role to perform well in this domain.

While sophisticated ordering schemes are out of scope of this work, we additionally evaluate a simple approach that orders action variables based on the name of the variable in either increasing (INC) or decreasing (DEC) order. The first two configurations in Table 1 show the results of the different orderings with h^{dc} as initialisation heuristic. On most domains the two orderings perform similarly; an increasing ordering is beneficial on ACADEMIC ADVISING, CHROMATIC DICE, EARTH OBSERVATION and WILDFIRE, while the decreasing ordering favours COOPERATIVE RECON, MANUFACTURER and RED-FINNED BLUE-EYE. We want to emphasise that the purpose of this experiment is to show that different orderings can have different impact, not that one of these arbitrary orderings is better than the other.

Finally, we evaluate the impact of mutex detection. The third algorithm in Table 1 is again based on an increasing ordering, but we perform mutex detection beforehand. It can be seen that mutex precomputation is almost always beneficial and increases the total IPC score by more than 20 points, since it has less overhead in the case where actions are not concurrently applicable.

Factored and Flattened Action Representation

In our second suite of experiments we compare the flattened representation to the factored representation with the factored-max heuristic h^{fm}, the decomposition heuristic h^{dc}, and the factored-random heuristic h^{fr}. The factored configurations precompute mutexes and order variables in increasing order; the reference action for h^{dc} is the action where 0 is assigned to all base actions. The four last columns of Table 1 show the IPC scores among all configurations over all domains and instances. In total, no configuration completely outperforms the baseline, although h^{dc} is almost as strong and the factored representation works quite well in CHROMATIC DICE. However, looking at the results among all instances is not necessarily an indicator for the performance of the factored representation, as this also includes instances where a factored representation is not particularly useful.

We therefore also report the results for problems that do not allow for concurrent actions, presented in the first two configurations in Table 2. In this case the factored representation with either heuristic represents the same algorithm as the flattened configuration does, and differences are only due to differences in the implementation. The difference in ACADEMIC ADVISING 2018 is traced back to a single instance where *flat* performs significantly better (avg. reward of -39.30 compared to -69.05 for h^{fm}) due to the IDS heuristic having a slightly higher search depth (3 instead of 2). This is also the case in MANUFACTURER, but here the increased depth harms search. In SYSADMIN there are exponentially many probabilistic outcomes. Our implementation performs less total trials, therefore the original implementation is more informed. In TRIANGLE TIREWORLD the mutex detection algorithm, which has quadratic run-time in the number of action variables, takes most of the total planning time.

Finally, the most interesting part of the benchmark set are problem instances which allow for concurrency and where a factored representation can potentially pay off. Table 3 depicts the IPC score of the different configurations for problems which allow for concurrency.

The total result as well as the results in CHROMATIC DICE, WILDLIFE PRESERVE and MANUFACTURER show

Domain	weight 0.5 $h^{\mathbf{dc}}$	weight 0.5 **flat**	weight 2 $h^{\mathbf{dc}}$	weight 2 **flat**
AA'14 (5)	**1.99**	1.98	**1.98**	**1.98**
AA'18 (5)	3.49	**4.65**	3.69	**4.59**
CROSS. (10)	9.34	**9.44**	7.69	**7.70**
ELEVATORS (4)	3.82	**3.90**	**3.87**	3.83
GAME (10)	**9.81**	**9.81**	9.61	**9.74**
LUCK (6)	5.48	**5.51**	5.82	**5.94**
MAN. (3)	**2.47**	1.70	**1.96**	1.24
NAV. (10)	**9.63**	9.42	**7.58**	7.54
RECON (10)	9.73	**9.93**	**9.97**	6.68
SKILL (10)	**9.86**	9.66	**9.73**	9.70
SYSADMIN (10)	8.01	**9.93**	8.54	**9.92**
TAMARISK (10)	9.57	**9.80**	9.43	**9.72**
TRIANGLE (10)	6.10	**7.42**	3.92	**5.78**
WILDFIRE (10)	**7.69**	7.45	6.25	**6.76**
WILDLIFE (7)	6.92	**6.98**	**8.68**	8.16
Sum (120)	103.91	**107.58**	98.71	**99.28**

Table 2: IPC scores for problems that do not allow concurrent actions for two different heuristic weights. Note that in this case $h^{\text{dc}} = h^{\text{fm}} = h^{\text{fr}}$.

Domain	$h^{\mathbf{dc}}$	$h^{\mathbf{fm}}$	$h^{\mathbf{fr}}$	**flat**
AA'14 (5)	0.97	0.98	0.83	**2.00**
AA'18 (15)	0.73	**1.00**	0.92	0.87
CHROM. (20)	**19.32**	18.18	19.20	16.99
COOP. RECON (20)	**12.07**	11.38	11.21	12.00
EARTH (20)	17.83	18.00	**19.19**	17.64
ELEVATORS (6)	5.78	5.85	5.09	**5.90**
LUCK (14)	7.57	10.98	8.99	**11.89**
MAN. (17)	3.45	2.67	**5.40**	2.41
RFBE (20)	12.19	12.31	11.35	**12.82**
TRAFFIC (10)	9.72	9.40	9.56	**9.85**
WILDLIFE (13)	8.49	8.09	**10.47**	3.91
Sum (160)	98.11	98.82	**102.20**	96.27

Table 3: IPC scores for problems that do allow for concurrent actions with a heuristic weight of 0.5.

Domain	$h^{\mathbf{dc}}$	$h^{\mathbf{fm}}$	$h^{\mathbf{fr}}$	**flat**
AA'14 (5)	4.82	**4.99**	0.72	1.00
AA'18 (15)	1.44	**3.00**	0.92	1.46
CHROM. (20)	**19.17**	18.57	17.76	15.54
COOP. RECON (20)	11.74	**14.18**	11.46	12.86
EARTH (20)	**18.91**	17.99	18.54	17.73
ELEVATORS (6)	5.49	5.37	**5.80**	5.41
LUCK (14)	8.58	9.72	9.22	**11.40**
MAN. (17)	**3.14**	1.48	2.00	0.65
RFBE (20)	10.39	10.32	11.14	**12.27**
TRAFFIC (10)	9.58	9.48	9.59	**9.91**
WILDLIFE (13)	7.64	7.68	**10.07**	3.50
Sum (160)	100.90	**102.80**	97.21	91.72

Table 4: IPC scores for problems that do allow for concurrent actions with a heuristic weight of 2.0.

that our heuristics successfully guide the factored THTS approach into relevant parts of the search space, and that the resulting planner is able to outperform a planner that works on the flat representation. Inspecting the results of ACADEMIC ADVISING'14 reveals that the difference is due to a single instance where the factored representation performs slightly worse than the default policy (-203 vs. -200) and the flattened configuration performs slightly better (-197.99 vs -200) and therefore the flattened representation gets a IPC score of 1, while the other configurations get a IPC score of 0 for this instance. In PUSH YOUR LUCK, the flattened representation significantly outperforms every factored configuration. We believe that this is partially due to the variable ordering, as a minor experiment that compares orderings based on the size of $\mathcal{D}_{\text{b}}$ indicates that such an ordering is more beneficial for this domain (results of other domains were not significantly different).

Comparing the different heuristics of the factored representation among each other shows that the factored-random heuristic, which is the fastest to compute, performs slightly better than the other two. However, the differences are small, and the number of applicable actions is rarely large enough that the theoretical advantage of the decomposed heuristic pays off in practice. However, if we slightly increase the weight the planner uses for the heuristic to 2 (the default configuration of the PROST planner multiplies the heuristic value with 0.5), we can already get a glimpse at the potential of the guidance of our new heuristics. Table 4 shows that both the decomposed heuristic and the factored-max heuristic achieve much better results in the 5 hard concurrent ACADEMIC ADVISING'14 instances than the flat representation does.

Conclusion

The presented theoretical and empirical evaluation shows that a factored representation can be beneficial for THTS algorithms and is an important first step towards efficient anytime optimal domain-independent algorithms for MDPs with exponentially many actions. However, more sophisticated approaches are required to tackle problems with exponentially large action spaces, as it is not feasible to iterate over all applicable actions anymore. The work of Raghavan (2017) who represents the action space as a decision diagram might be a promising idea. Our evaluation on action orderings only considered static orderings; dynamically adapting the ordering might be a beneficial alternative for many domains, as a good action ordering most likely depends on the current state, and an interesting future direction is to base the ordering on initial heuristic estimates.

We believe that the presented framework of factored action spaces coupled with dynamic variable orderings and fast and efficient consistency checks can be the key to solving even the most challenging problems of the IPPC 2018 benchmarks, and will thus be beneficial for a broad range of application domains as well.

Acknowledgements. We thank Reviewer 1 for the helpful comments and suggestions. Florian Geißer was supported

by ARC project DP180103446, "On-line Planning for Constrained Autonomous Agents in an Uncertain World". David Speck was supported by the German National Science Foundation (DFG) as part of the project EPSDAC (MA 7790/1-1). Thomas Keller received funding for this work from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 817639).

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine Learning* 47:235–256.

Bellman, R. E. 1957. *Dynamic Programming*. Princeton University Press.

Bubeck, S.; Munos, R.; and Stoltz, G. 2009. Pure exploration in multi-armed bandits problems. In *ALT*, volume 5809 of *Lecture Notes in Computer Science*, 23–37.

Chaslot, G.; Winands, M. H. M.; Van Den Herik, H. J.; Uiterwijk, J. W. H.; and Bouzy, B. 2008. Progressive strategies for monte-carlo tree search. *New Mathematics and Natural Computation* 04:343–357.

Geißer, F., and Speck, D. 2018. Prost-DD – utilizing symbolic classical planning in THTS. In *Sixth International Probabilistic Planning Competition (IPC-6): planner abstracts*, 13–16.

Geißer, F.; Speck, D.; and Keller, T. 2019. An analysis of the probabilistic track of the IPC 2018. In *ICAPS 2019 Workshop on the International Planning Competition (WIPC)*, 27–35.

Guestrin, C.; Koller, D.; Parr, R.; and Venkataraman, S. 2003. Efficient solution algorithms for factored MDPs. *Journal of Artificial Intelligence Research* 19:399–468.

Helmert, M. 2009. Concise finite-domain representations for PDDL planning tasks. *Artificial Intelligence* 173:503–535.

Keller, T., and Eyerich, P. 2012. PROST: Probabilistic planning based on UCT. In McCluskey, L.; Williams, B.; Silva, J. R.; and Bonet, B., eds., *Proceedings of the Twenty-Second International Conference on Automated Planning and Scheduling (ICAPS 2012)*, 119–127. AAAI Press.

Keller, T., and Helmert, M. 2013. Trial-based heuristic tree search for finite horizon MDPs. In Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds., *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling (ICAPS 2013)*, 135–143. AAAI Press.

Keller, T. 2015. *Anytime Optimal MDP Planning with Trial-based Heuristic Tree Search*. Ph.D. Dissertation, University of Freiburg.

Kocsis, L., and Szepesvári, C. 2006. Bandit based Monte-Carlo planning. In Fürnkranz, J.; Scheffer, T.; and Spiliopoulou, M., eds., *Proceedings of the 17th European Conference on Machine Learning (ECML 2006)*, volume 4212 of *Lecture Notes in Computer Science*, 282–293. Springer-Verlag.

Koller, D., and Parr, R. 1999. Computing factored value functions for policies in structured MDPs. In Dean, T., ed., *Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence (IJCAI 1999)*, 1332–1339. Morgan Kaufmann.

Moraes, R. O.; Mariño, J. R. H.; Lelis, L. H. S.; and Nascimento, M. A. 2018. Action abstractions for combinatorial multi-armed bandit tree search. In *Proceedings of the Fourteenth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE 2018)*, 74–80.

Nilsson, N. J. 1980. *Principles of Artificial Intelligence*. Morgan Kaufmann.

Ontañón, S. 2013. The combinatorial multi-armed bandit problem and its application to real-time strategy games. In desJardins, M., and Littman, M. L., eds., *Proceedings of the Twenty-Seventh AAAI Conference on Artificial Intelligence (AAAI 2013)*. AAAI Press.

Pommerening, F.; Helmert, M.; Röger, G.; and Seipp, J. 2015. From non-negative to general operator cost partitioning. In Bonet, B., and Koenig, S., eds., *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI 2015)*, 3335–3341. AAAI Press.

Povéda, G.; Regnier-Coudert, O.; Teichteil-Königsbuch, F.; Dupont, G.; Arnold, A.; Guerra, J.; and Picard, M. 2019. Evolutionary approaches to dynamic earth observation satellites mission planning under uncertainty. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*, 1302–1310.

Puterman, M. L. 1994. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc.

Raghavan, A. 2017. *Domain-Independent Planning for Markov Decision Processes with Factored State and Action Spaces*. Ph.D. Dissertation, Oregon State University.

Reyes, A.; Ibargüengoytia, P. H.; Romero, I.; Pech, D.; and Borunda, M. 2015. Open questions for building optimal operation policies for dam management using factored markov decision processess. In *Sequential Decision Making for Intelligent Agents: Papers from the AAAI Fall Symposium*, 69–74. AAAI Press.

Sanner, S. 2010. Relational dynamic influence diagram language (RDDL): Language description.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of Go without human knowledge. *Nature* 550(7676):354–359.

Xue, S.; Fern, A.; and Sheldon, D. 2014. Dynamic resource allocation for optimizing population diffusion. In *Proceedings of the Seventeenth International Conference on Artificial Intelligence and Statistics (AISTATS 2014)*, 1033–1041. JMLR.org.